Modeling and Verification of Real-Time Software Using Extended Linear Hybrid Automata

Steve Vestal
steve.vestal@honeywell.com
Honeywell Technology Center
Minneapolis, MN 55418*

Abstract

Linear hybrid automata are finite state automata augmented with real-valued variables. Transitions between discrete states may be conditional on the values of these variables and may assign new values to variables. These variables can be used to model real time and accumulated task compute time as well as program variables. Although it is possible to encode preemptive fixed priority scheduling using existing linear hybrid automata models, we found it more general and efficient to extend the model slightly to include resource allocation and scheduling semantics. Under reasonable pragmatic restrictions for this problem domain, the reachability problem is decidable. The proof of this establishes an equivalence between dense time and discrete time models given these restrictions. We next developed a new reachability algorithm that significantly increased the size of problem we could analyze, based on benchmarking exercises we carried out using randomly generated real-time uniprocessor workloads. Finally, we assessed the practical applicability of these new methods by generating and analyzing hybrid automata models for the core scheduling modules of an existing real-time executive. This exercise demonstrated the applicability of the technology to real-world problems, detecting several errors in the executive code in the process. We discuss some of the strengths and limitations of these methods and possible future developments that might address some of the limitations.

1 Introduction

The first goal of the work described in this paper was to analyze the schedulability of real-time systems that cannot be easily modeled using traditional scheduling theory. Traditional real-time task models cannot easily handle variability and uncertainty in clock and computation and communication times, synchronizations (rendezvous) between tasks, remote procedure calls, anomalous scheduling in distributed systems, dynamic reconfiguration and reallocation, end-to-end deadlines, and timeouts and other error handling behaviors.

*This work has been supported by the Air Force Office of Scientific Research under contract F49620-97-C-0008.

The second goal was to verify software implementations of systems. Task schedulers and communications protocols are reactive components that respond to events like interrupts, service calls, task completions, error detections, etc. We would like to model important implementation details such as control logic and data variables in the code. We would like the mapping between model and code to be clear and simple to better assure that the model really does describe the implementation.

Discrete event concurrent process models are widely used to model control flow within and interactions between concurrent activities. Classical discrete event concurrent process models do not deal with resource allocation and scheduling or data variables, which limits their usefulness for real-time systems and makes it awkward to model some implementation details. Classical preemptive scheduling models do not deal well with complex task sequencing and interaction, which limits their usefulness for describing distributed systems and implementation details. Discrete time models have been developed for real-time scheduling of concurrent processes[23, 13, 11, 31], and some work has been done on dense time real-time process algebras[10, 14]. This paper describes the use of dense time linear hybrid automata models to perform schedulability analysis and to verify implementation code.

The first problem we faced was the modeling of resource allocation and scheduling behaviors using hybrid automata. The applicability in principle of hybrid automata to the scheduling problem was already known[4]. We wanted a model that would admit a variety of complex allocation as well as scheduling algorithms, e.g. load balancing, priority inheritance. We wanted to be able to change the allocation and scheduling algorithms easily without changing the models of the real-time tasks themselves. We wanted to minimize the number of states and variables added to model allocation and scheduling. We found it most general and efficient to extend the definition of hybrid automata to include resource allocation and scheduling semantics rather than try to model the scheduling function as a hybrid automaton.

We use integration variables to record the accumulated compute time of tasks in preemptively sched-

uled systems. Allowing integration variables is known to make the reachability problem undecidable[22, 17]. We were curious about whether analysis of real-time allocation and scheduling in distributed heterogeneous systems is itself a fundamentally difficult problem, or if general linear hybrid automata are more powerful than is really necessary for this problem. We were able to show that the reachability problem becomes decidable when some simple pragmatic restrictions are placed on the model.

The second problem we faced was the computational difficulty of performing a reachability analysis. We began our work using an existing linear hybrid automata analysis tool, HyTech[18], but found ourselves limited to very small models. We developed and implemented a new reachability method that was significantly faster, more numerically robust, and used less memory. However, our prototype tool allows only constant rates (not rate ranges) and does not provide parametric analysis.

Using this new reachability procedure we were able to accomplish one of our goals: the modeling and verification of a piece of real-time software. We developed a hybrid automata model for that portion of the MetaH real-time executive that implements uniprocessor task scheduling, time partitioning and error handling[1]. We successfully analyzed these models, uncovering several implementation defects in the process. There are limits on the degree of assurance that can be provided, but in our judgement the approach may be significantly more thorough and significantly less expensive that traditional testing methods. This result suggests the technology has reached the threshold of practical utility for the verification of small amounts of software of a particular type.

However, we do not believe existing reachability methods are adequate yet for schedulability analysis of real systems. In our judgement, we would need to be able to analyze systems having a few dozen tasks on a few processors in order for the technology to begin finding use in this area. We discuss approaches that might lead to such improvements.

2 Resourceful Hybrid Automata

A hybrid automaton is a finite state machine augmented with a set of real-valued variables and a set of propositions about the values of those variables. Figure 1 shows an example of a hybrid automaton whose discrete states are `preempted, executing` and `waiting`; and whose real-valued variables are c and t. `Waiting` is marked as the initial discrete state, and c and t are assumed to be initially zero.

Each of the discrete states has an associated set of differential equations, e.g. $\dot{c} = 0$ and $\dot{t} = 1$ for the discrete state `preempted`. While the automaton is in a discrete state, the continuous variables change at the rates specified for that state.

Edges may be labeled with guards involving continuous variables, and a discrete transition can only occur when the values of the continuous variables satisfy the guard. When a discrete transition does occur, designated continuous variables can be set to designated values as specified by assignments labeling that edge.

A discrete state may also be annotated with an invariant constraint to assure progress. Some discrete transition must be taken from a state before that state's invariant becomes false. For example, the hybrid automaton in Figure 1 must transition out of state `computing` before the value of c exceeds 100.

The hybrid automata of interest to us are called linear hybrid automata because the invariants, guards and assignments are all expressed as sets of linear constraints. The differential equations governing the continuous dynamics in a particular discrete state are restricted to the form $\dot{x} \in [l, u]$ where $[l, u]$ is a fixed constant interval (our current prototype tool is further restricted to a singleton rate, $\dot{x} = [l, l]$).

We want to verify assertions about the behavior of a hybrid automaton. Although it is possible in general to check temporal logic assertions[4], we make do by annotating discrete states and edges with sets of linear constraints labeled as assertions. These constraints must be true whenever the system is in a discrete state or whenever a transition occurs over an edge.

The cross-product construction used to compose concurrent finite state processes can be extended in a fairly straight-forward way to systems of hybrid automata. The invariant and assertion associated with a discrete system state are the conjunction of the invariants and assertions of the individual discrete states. The guards, assertions and assignments of synchronized transitions are the conjunction and union of the guards, assertions and assignments of the individual discrete co-edges. If there is a conflict between the rate assignments of individual discrete states, or a conflict between the variable assignments of co-edges, then the system is considered ill-formed. Note that concurrent hybrid automata may interact through shared real-valued variables as well as by synchronizing their transitions over co-edges.

The application of interest in this paper is the analysis and verification of real-time systems. Figure 1 shows an example of a simple hybrid automata model for a preemptively scheduled, periodically dispatched task. A task is initially waiting for dispatch but may at various times also be executing or preempted. The variable t is used as a timer to control dispatching and to measure deadlines. The variable t is set to 0 at each dispatch (each transition out of the waiting state), and a subsequent dispatch will occur when t reaches 1000. The assertion $t \leq 750$ each time a task transitions from executing to waiting (each time a task completes) models a task deadline of 750 time units. The variable c records accumulated compute time, it is reset at each dispatch and increases only when the task is in the computing state. The invariant $c \leq 100$ in the computing state means the task must complete before it receives more than 100 time units of processor service, the guard $c \geq 75$ on the completion transition means the task may complete after it has received 75 time units of processor service (i.e. the task compute time is uncertain and/or variable but always falls in the interval $[75, 100]$).

In this example the edge guards `selected` and `unselected` represent scheduling decisions made at

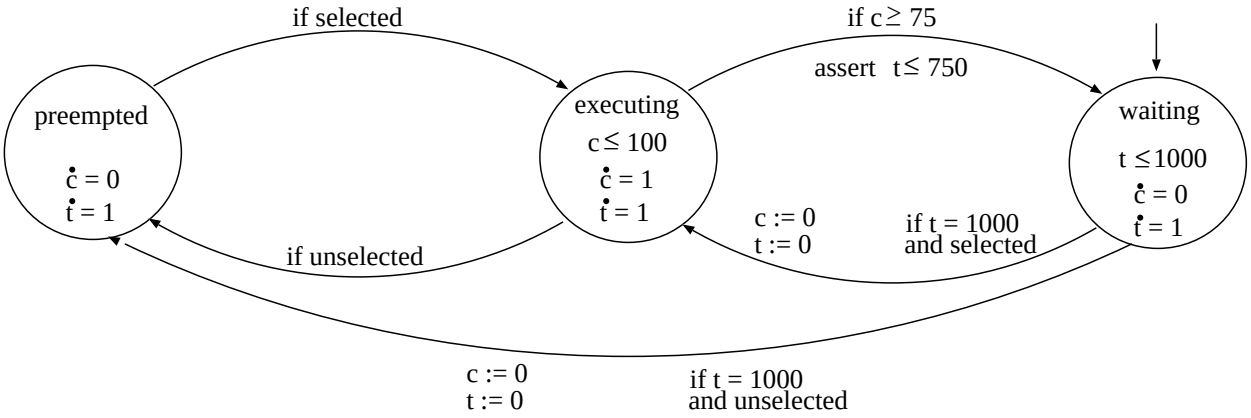


Figure 1: A Hybrid Automata Model of a Preemptively Scheduled Task

scheduling events (called scheduling points in the real-time literature). These decisions depend on the available resources (processors, busses, etc.) being shared by the tasks. There are several approaches to introduce scheduling semantics into a model having several concurrent tasks.

Scheduling can be introduced using concepts taken from the theory of discrete event control[26]. A concurrent scheduler automaton can be added to the system of tasks. The scheduling points in the task set become synchronization events at which the scheduler automaton can observe the system state and make control decisions. Many high-level concepts from discrete event control theory carry over into this domain, such as the importance of decentralized control and limited observability in distributed systems.

Discrete event control theory provides an approach to synthesize optimal controllers, which in this domain translates to the automatic construction of application-specific scheduling algorithms. However, classical discrete event control theory does not deal with time. The theory has been extended to synthesize nonpreemptive schedulers for timed automata[9, 2], but this excludes preemptively scheduled systems. It is possible to develop scheduling automata by hand using traditional real-time scheduling policies such as preemptive fixed priority. Some examples have been given in the literature, where each distinct ready queue state is modeled as a distinct discrete state of the scheduler automaton[4]. This would allow a very large class of scheduling algorithms to be modeled, but the size of the scheduler automaton may grow combinatorially with the number of tasks.

It is possible to model preemptive fixed priority scheduling by encoding the ready queue in a variable rather than in a set of discrete states. A queue variable is introduced that will take on only integer values. At each transition where a task i is dispatched, 2^i is added to this queue variable; at each transition where task i completes, 2^i is subtracted. The queue variable can be interpreted as a bit vector whose i^{th} bit is set whenever task i is ready to compute. There is no separate scheduler automaton, the scheduling protocol is modeled using additional guards and states in the task automata. This is the approach we took when we started our work using HyTech. This encodes a specific scheduling protocol into each task model, and adds additional discrete states, variables and guards to the model. It is awkward to model any scheduling policy other than simple preemptive fixed priority.

In the end, we found it simpler and more general to define a slightly extended linear hybrid automata model that includes resource scheduling semantics[28]. The discrete state composition of the task set is performed before any scheduling decisions are made. A scheduling function is then applied to the composed system discrete state to determine the variable rates to be used for that system state. In essence, the composed system discrete state is the ready queue to which the scheduling function is applied, very much analogous to the way run-time scheduling algorithms are applied in an actual real-time system. It is not necessary to have different discrete states for preempted and computing, since this information is now captured in the variable rates. It is not necessary to model a scheduling algorithm as a finite state control automaton added to the system, it is not necessary to encode a specific scheduling semantics into the task automata. One simply codes up a scheduling algorithm in the usual way and links it with the rest of the reachability analysis code. This approach significantly reduces the number of discrete states in the model (from 3^t for our HyTech models to 2^t for our extended models, where t is the number of tasks). This also simplifies the modeling of the desired scheduling discipline. The details of this model and its semantics are recorded elsewhere[28].

3 Decideability

Most traditional real-time schedulability problems are solvable in polynomial time or are NP-complete. However, hybrid automata models that allow multiple rates and integration variables are undecideable[22, 17]. The hybrid automata models we are using are much more powerful than traditional allocation and

scheduling models, and most existing tasking and scheduling models can be viewed as special cases of the more general hybrid automata model. This raises the question of whether the schedulability problem for complex interacting tasks that are dynamically allocated in distributed heterogeneous systems is in fact undecideable, or whether models of such systems are decideable special cases of the more powerful linear hybrid automata models.

The undecideability of hybrid automata reachability analysis was proved by reducing the reachability problem for two-counter machines, which is known to be undecideable, to the reachability problem for hybrid automata[22, 17]. The construction used in the proof is fairly straightforward in our slightly extended model and can be accomplished using a single processor. However, a pragmatic real-time system designer would reject the theoretical construction as a bad design because it relies in places on exact equality comparisons between timers and accumulated compute times. In a real system, these would be regarded as race conditions or ill-defined behaviors. The problem becomes decideable given a few simple practical restrictions, which are captured in the following theorem.

Theorem 1 *The reachability problem is decideable for resourceful linear hybrid automata if the following conditions hold.*

- *The set of possible outputs of the scheduling function for each possible system discrete state is finite and enumerable.*
- *For every task activity integrator variable, the rate interval remains fixed between resets of that integrator (i.e. the scheduler does not dynamically reallocate any task activity in mid-execution to a new resource having a different rate for that activity).*
- *For every task activity integrator variable, every edge guard is a set of rectangular constraints of the form $x \in [l, u]$, and either the edge guard has a non-singular interval ($x \in [l, u]$ with $l < u$) or else the rate interval for $\dot{x}$ is non-singular (i.e. system behavior does not depend on exact equality comparisons with exact drift-free clocks or execution rates).*
- *However, we allow as a special exception task activity integrator variables with singular rate interval and singular rectangular edge guards, providing the integrator variable is only reset or stopped or restarted at a transition having at least one edge guard $y \in [m, m]$ with $[m, m]$ and $\dot{y}$ singular (y may but need not be x), and for every such singular constraint on that edge $\dot{x} = k\dot{y}$ for some positive integer k (i.e. some types of noninteracting or harmonically interacting behaviors may be modeled exactly).*

This result should not be surprising. The ability to test for exact equality is known to add theoretical power to dense time temporal logics[3], and similar restrictions are known to make certain other hybrid automata models decideable[25]. The proof of this theorem, which we provide elsewhere[28], is by reduction to a discrete time finite state automaton.

4 Reachability Analysis

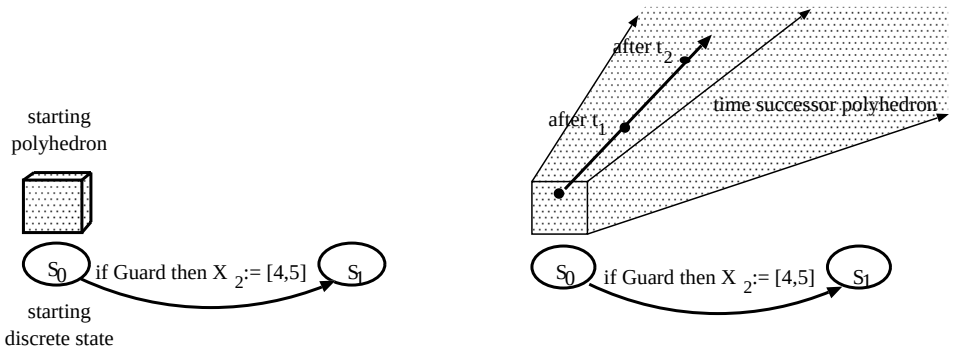
A state of a linear hybrid automaton consists of a discrete part, the discrete state at some time t; and a continuous part, the real values of the variables at time t. It turns out that, although this state space is uncountably infinite, the reachable state space for a given linear hybrid automaton is a subset of the cross-product of the discrete states with a recursively enumerable set of convex polyhedra in $\Re^n$ (where n is the number of variables)[4]. A region of a linear hybrid automaton is a pair consisting of a discrete state and a convex polyhedron, where convex polyhedra can be represented using a finite set of linear constraints. Model checking consists of enumerating the reachable regions for a given linear hybrid automaton and checking to see if they satisfy the assertions.

Figure 2 depicts the basic sequence of operations that, given a starting region (a discrete state and a polyhedron defining a set of possible values for the variables), computes the set of values the variables might take on in that discrete state as time passes; and computes a set of regions reachable by subsequent discrete transitions.

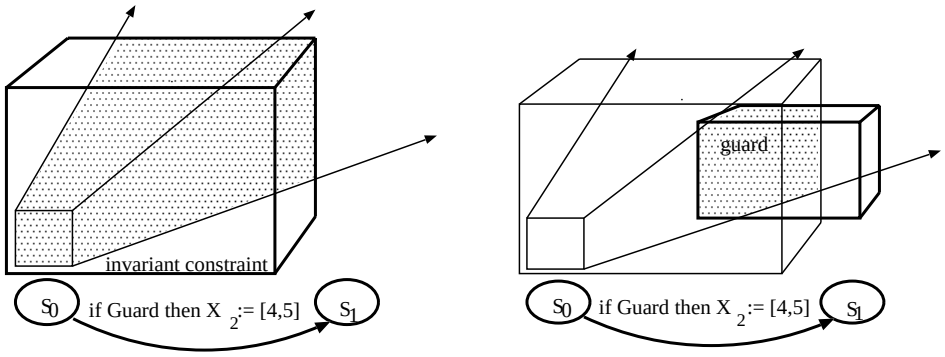
The first step is the computation of the time successor polyhedron from the starting polyhedron (often called the post operation). For each point in the starting polyhedron, the time successor of that point is a line segment beginning at that point whose slope is defined by the variable rates specified for the discrete state. This is the set of variable values that can be reached from a starting point by allowing some amount of time to pass. The time successor of the starting polyhedron is the union of the time successor lines for all points in the starting polyhedron. A basic result of linear hybrid automata theory is that the time successor of any convex polyhedron is itself a convex polyhedron (which in general will be unbounded in certain directions)[4].

The second step is the intersection of the time successor polyhedron with the invariant constraint associated with the discrete state. Polyhedra are easily intersected by taking the union of the set of linear constraints that define the two polyhedra. This is the time successor region that is feasible given the invariant specified for the discrete state.

The remaining steps are used to compute new regions reachable from this feasible time successor region by some transition over an edge. For each edge out of the current discrete state, the associated guard is first intersected with the feasible time successor region. This polyhedron, if nonempty, defines the set of all variable values that might exist whenever the discrete transition could occur. Any variable assignments associated with the edge must now be applied to this polyhedron. This is done in two phases. First, a variable to be assigned a new value $x := l$ is unconstrained (often called the free operation). This oper-

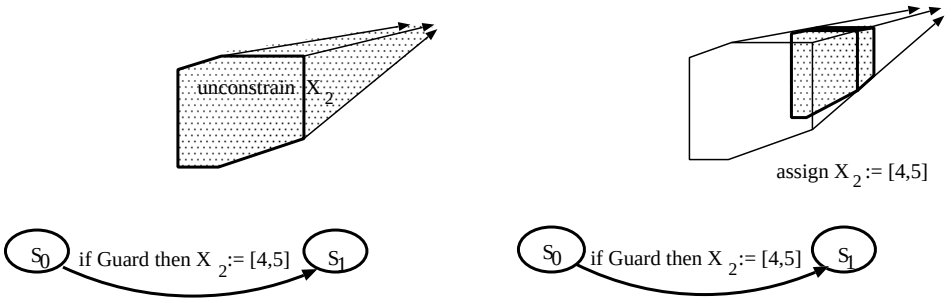


Step 1: Time Successor



Step 2: Intersection with Invariant

Step 3: Intersection with Guard



Step 4: Unconstrain Assigned Variable

Step 5: Intersect with Assignment Constraint

Figure 2: Hybrid Automata Reach Forward Operations

ation leaves unchanged the relationships between all other variables, i.e. the polyhedron is projected onto the subspace $\Re^{n-1}$ of the remaining variables. This result is then intersected with the constraint $x = l$. This polyhedron, together with the discrete state to which the edge goes, is a new region for which the above steps may be repeated. In general a set of assignments whose right-hand sides are linear formula are allowed, with some restrictions. The variables to be assigned are unconstrained and the resulting polyhedra are then intersected with the appropriate linear constraints in some order. With care, fairly complex

sequences of assignments to program variables can be modeled on a single edge[30].

The overall method begins at the initial region of a hybrid automaton. The operations described above are applied to enumerate feasible time successor regions and the new regions reachable from these via discrete transitions. As new regions are enumerated, they must be checked to see if they have been visited before (otherwise the method will not terminate even when there are a finite number of regions). This is done by comparing the discrete states of regions for equality, and by checking to see if the new polyhedron is contained in the polyhedron of a previously visited region.

The earliest reachability tool of which we are aware, HyTech, represented polyhedra as finite sets of linear constraints[4]. Operations on polyhedra used quantifier elimination, a method to manipulate and make decisions about systems of linear constraints in which some of the variables are existentially quantified. Subsequent tools, Polka and a later version of HyTech, used a pair of representations: the traditional system of linear constraints together with polyhedra generators consisting of sets of vertices and rays[16, 18]. Different operations required during reachability are more convenient in the different representations, and methods are used to convert between the two as needed.

Both of these methods are subject to the theoretical risk that some polyhedra operations may require a combinatorial amount of time. Another potential performance problem occurs when the reachable discrete state space is completely enumerated first followed by an enumeration of the polyhedra. This might result in enumerating discrete states that are actually not reachable due to edge guards involving the continuous variables. Finally, in our experiments we found that a significant fraction of a set of benchmark schedulability problems we tried to solve using HyTech resulted in numeric overflow errors.

We developed a new set of algorithms for the polyhedra operations used during reachability analysis and implemented a prototype on-the-fly reachability analysis library. Our prototype operates on lists of linear constraints of the form $l \leq e \leq u$ where l and u are fixed constant integer bounds and $e = c_1x_1 + c_2x_2 + ...$ is a linear formula with fixed constant integer coefficients. Our current algorithms restrict variable rates to be fixed scalar constants, $\dot{x} = i$ rather than the more general $\dot{x} \in [l, u]$.

We convert a polyhedron P into $\text{Post}(P, \dot{\mathbf{x}})$, the time successor of P given a vector of variable rates $\dot{\mathbf{x}}$, by applying the two steps

1. Let each constraint $l_i \leq e_i \leq u_i$ where $\dot{e_i} \neq 0$ be written so that $\dot{e_i} > 0$, which can be achieved by multiplying the constraint by -1 if needed. For each distinct pair of constraints

$$l_i \leq e_i \leq u_i$$
$$l_j \leq e_j \leq u_j$$

   where $\dot{e_i} > 0$ and $\dot{e_j} > 0$, add to the set the constraint

$$\dot{e_j}l_i - \dot{e_i}u_j \leq \dot{e_j}e_i - \dot{e_i}e_j \leq \dot{e_j}u_i - \dot{e_i}l_j$$

2. Replace each constraint $l \leq e \leq u$ where $\dot{e} > 0$ by $l \leq e \leq \infty$.

We compute $\text{Free}(P, x)$, the result of unconstraining variable x in polyhedron P, using the two steps

1. Let each constraint $l \leq e \leq u$ in P where e has an instance of x be written in the form $l \leq cx - e' \leq u$, where e' involves the remaining variables and their coefficients and $c > 0$. For every distinct pair of such constraints in P

$$l_i \leq c_ix - e_i \leq u_i$$
$$l_j \leq c_jx - e_j \leq u_j$$

   combine the two in a way that cancels the x terms, adding to $\text{Free}(P, x)$ the constraint

$$c_jl_i - c_iu_j \leq c_ie_j - c_je_i \leq c_ju_i - c_il_j$$

2. Each constraint $l \leq e \leq u$ where e has no instances of variable x is added to $\text{Free}(P, x)$.

These algorithms might be viewed as generalizations of the difference methods used for timed automata[12, 8] and exhibit some similarity to the pragmatic algorithm used earlier for quantifier elimination[4]. Our prototype invokes a Simplex algorithm as part of the operations to test for feasibility and containment. We use a bounds tightening procedure to reduce the size of the constraint list after certain operations and to rapidly detect most infeasible polyhedra. Simplex-based reduction and feasibility testing is only applied when the bounds tightening procedure is ineffective. Details of our reachability analysis methods and implementation and proofs of correctness are documented elsewhere[29].

We benchmarked our prototype tool against HyTech and Verus[11] (a discrete timed automata reachability analysis tool that uses BDD techniques) using randomly generated uniprocessor workloads containing mixtures of periodic and aperiodic tasks. Figure 3 shows the percentage of problems that were solved by each of the tools, together with the primary reasons that solution was not achieved. Figure 4 compares the time required for solution for problems that were solved by all the tools using a logarithmic scale (a point appears for both HyTech and our prototype only for problems that were solved by both). We further increased the size of model we could analyze by applying some results from traditional scheduling theory to simplify the models, and by using a simple partial order reduction technique, these results are reported elsewhere[29].

5 Verifying the MetaH Executive

MetaH is an emerging SAE standard language for specifying real-time fault-tolerant high assurance software and hardware architectures[1, 24, 27]. Users

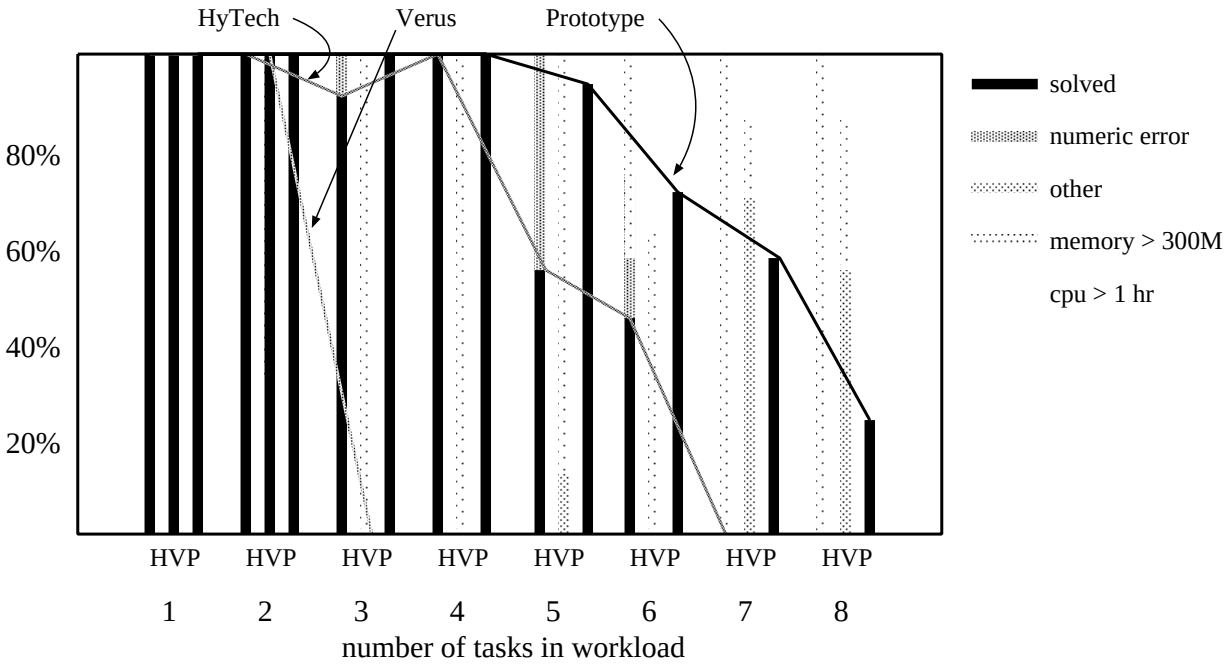


Figure 3: Percentage of Generated Problems That Were Solved

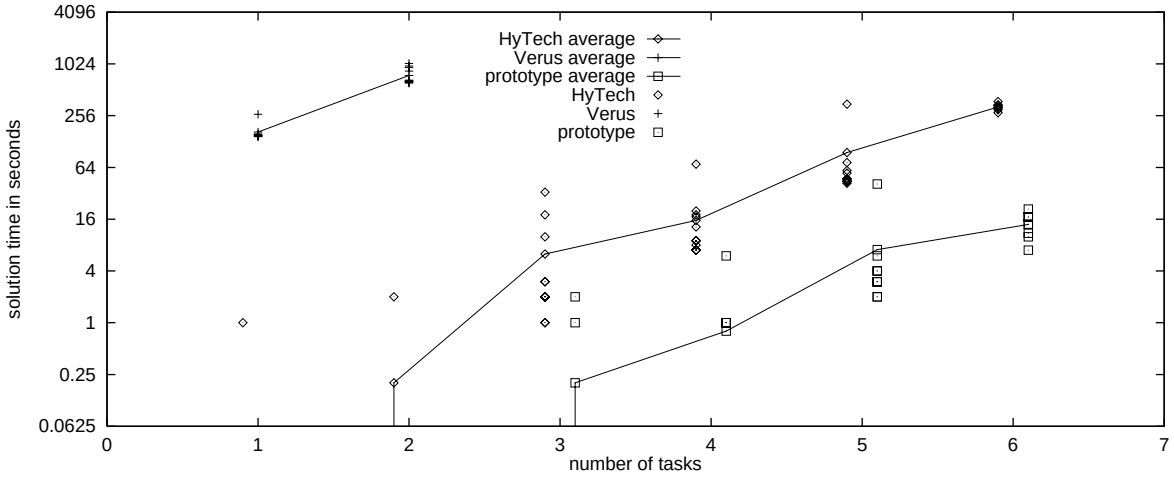


Figure 4: Solution Times for Problems That Were Solved

specify how software and hardware components are combined to form an overall system architecture. This specification includes information about one or more configurations of tasks and message and event connections; and information about how these objects are mapped onto a specified hardware architecture. The specification includes information about timing behaviors and requirements, fault and error behaviors and requirements, and partitioning and safety behaviors and requirements.

Our current MetaH toolset, illustrated in Figure 5, can generate and analyze formal models for schedulability, reliability, and partition isolation. The toolset can also configure an application-specific executive to perform the specified task dispatching and scheduling, message and event passing, changes between alternative configurations, etc. Unlike many conventional systems that rely on a large number of run-time service calls to configure a system by dynamically creating and linking to tasks, mailboxes, event channels, timers, etc., our toolset builds most of this information into an application-specific executive. There are relatively few run-time service calls, and the effects of these are tailored based on the specified application

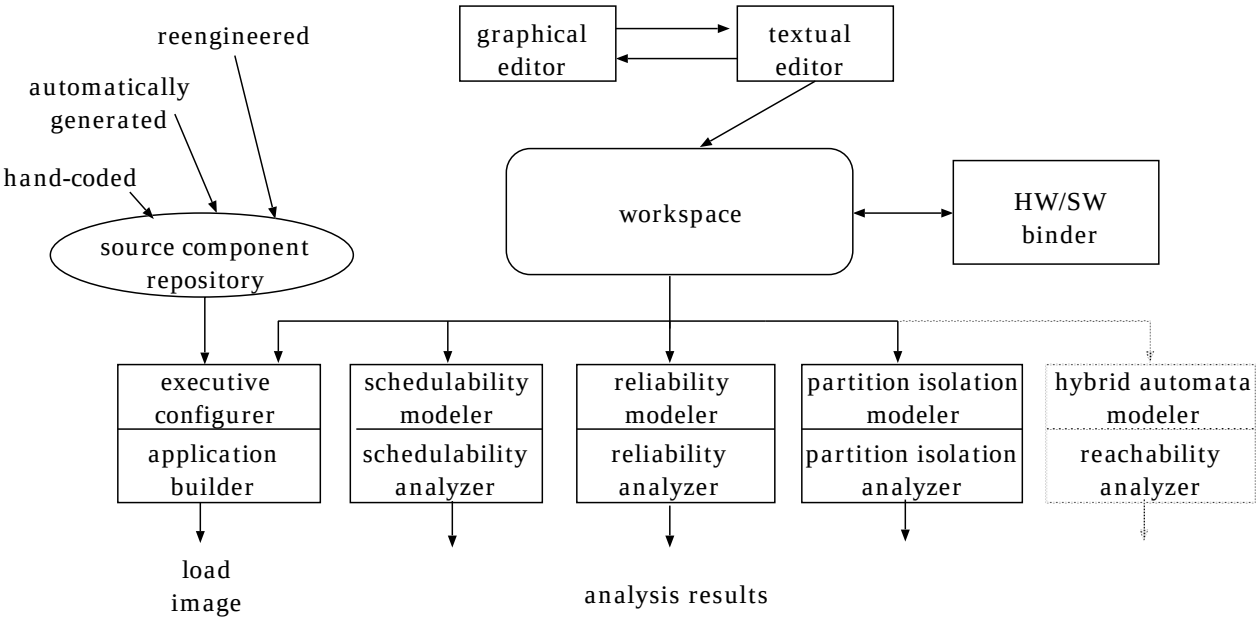


Figure 5: MetaH Toolset

architecture and requirements.

Our MetaH executive supports a reasonably complex tasking model using preemptive fixed priority scheduling theory[5, 6, 7]. Among the features relevant to this study are period-enforced aperiodic tasks, real-time semaphores, mechanisms for tasks to initialize themselves and to recover from internal faults, and the ability to enforce execution time limits on all these features (time partitioning). Slack stealing in support of aperiodic and incremental tasks is also supported, but as we will mention later these were not modeled or verified.

Figure 6 shows the high-level structure of the MetaH executive. The core task scheduling operations are implemented by module `Threads`, e.g. start, dispatch, complete. These operations implement transitions between the discrete task scheduling states, e.g. dispatch may transition a task from the awaiting dispatch state to the computing state. These operations must take into account details such as the task type, optional execution time enforcement, event queueing, etc. Module `Threads` invokes operations in module `Time_Slice`, which encapsulates arithmetic operations and tests on two execution time accumulators maintained by the underlying RTOS and hardware for each task: an accumulator that increases while a task executes, and a time slice that decreases while a task executes. `Time_Slice` may set these variables to desired values using services provided through the `MetaH RTOS interface`. If time slicing is enabled for a task, then a trap will be raised by the underlying hardware and RTOS when the time slice reaches zero. This trap is handled by one of the operations in `Threads`. Module `Clock_Handler` is periodically invoked by the underlying system (it is the handler for a periodic clock interrupt) and makes calls to `Threads` to dispatch periodic tasks and start and stop threads at mode changes. Modules `Events`, `Modes` and `Semaphores` contain data tables and operations to manage user-declared events, dynamic reconfiguration, and semaphores.

We produced hybrid automata models for the `Threads` and `Time_Slice` modules, about 1800 lines of code. We did not write a separate model using a special modeling language, instead we inserted calls to build the model into the executive code itself. For example, in the code that implements the dispatch operation there is logic to decide if a task can be dispatched, assignments to change program variables, and calls to set the time slice and execution time counters. Into this code we inserted a call to a modeling procedure to create an edge between the corresponding states of the linear hybrid automata model. The guards for this edge are the conditional expressions appearing in the code, and the assignments on this edge are the assignments appearing in the code. This provides a high degree of traceability between the implementation and the model.

The generation of the hybrid automata models resembled all-paths unit testing. We developed several simple application specifications that included most (but not all) of the tasking features. We wrote a test driver that exercised all relevant paths in the core scheduling modules. For each application specification, the test driver thus triggered the generation of a linear hybrid automata model of the possible behaviors of the core scheduling operations for a particular combination of tasks and features.

The conditions we checked during reachability analysis were that all deadlines were met whenever

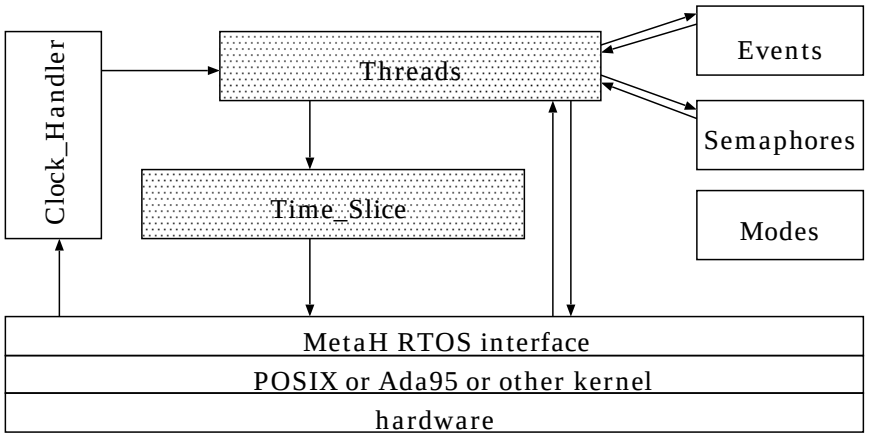


Figure 6: MetaH Executive Structure

the schedulability analyzer said an application was schedulable; no accessed variables were unconstrained (undefined) and no invariants were violated on entry to a region; and no two tasks were ever in a semaphore locking state simultaneously. Assertion checks appearing in the code were modeled by edges annotated with `assert False`.

We also collected information about which edges were used by some transition during reachability analysis and compared this with all the possible edges that might be created (all instances of calls inserted into the code to create edges). This allowed us to insure that all modeled portions of the code were covered by at least one reachability analysis.

A total of 14 real-valued variables and 15 discrete states were defined to model each task. No single task model used all 14 variables and 15 states, different task types with different specified options used different combinations. Figure 7 shows the simplest linear hybrid automata model we generated, a periodic task with period and deadline of 100000us, compute time between 0 and 90000us, recovery time between 0 and 10000us. States are also annotated with processor scheduling priorities, which are not shown here. The variable rates were derived from the scheduling priorities by the analysis tool, which used preemptive fixed priority scheduling semantics for this study. Table 1 summarizes the complete set of applications we analyzed. A more detailed discussion of the modeling methods and results is provided elsewhere[30].

We discovered nine defects in the course of our verification exercise. Four of these were tool defects, two that could cause bad configuration data to be generated and two that could cause erroneously optimistic schedulability models to be generated. Six of these defects could cause errors only during the handling of application faults and recoveries, three of these six only in the presence of multiple near-coincident faults and recoveries. In our judgement, of the nine defects we found, one would almost certainly have been detected by moderately thorough requirements testing, while three would have been almost impossible to detect by testing due to the multiple carefully timed events required to produce erroneous behavior. The other five may have been detected by thorough requirements testing of fault and recovery features, providing the tester thought about possible execution timelines and arranged for tasks to consume carefully selected amounts of time between events.

There are a number of significant limitations on the degree of assurance provided. In our initial exercise, we chose not to model many behaviors that could have been modeled in a fairly straight-forward way, e.g. mode changes, inter-processor communication protocol, non-preemptable executive critical sections. In some cases different behaviors and subsystems can be modeled and analyzed almost independently, but it is not clear at what point the reachability analysis will become intractable as the extent of the model grows. Some behaviors might be more difficult to model, e.g. slack scheduling. The MetaH processor interface, underlying RTOS and hardware are unlikely to be fully model-able for a variety of practical and technical reasons. The MetaH tools were not verified, only a few specific generated modules and reports for a few example applications. Although our approach provides good traceability between code and model, there is still a very real possibility of modeling errors. The reachability analysis tool may contain defects; we discovered two in our tool in the course of this work. The modeled code does not change from application to application, and the analyzed applications fully exercised the code model, but to rigorously assert this code is correct for all possible applications would require some sort of induction argument. Even if the source code is correct, defects in the compiler, linker or loader software could introduce defects into the executable image.

Nevertheless, we estimate that the effort required for this exercise was roughly comparable to that required for traditional unit testing, but the results were more thorough than would have been achieved using traditional requirements testing. The method must be used in conjunction with traditional verification tech-

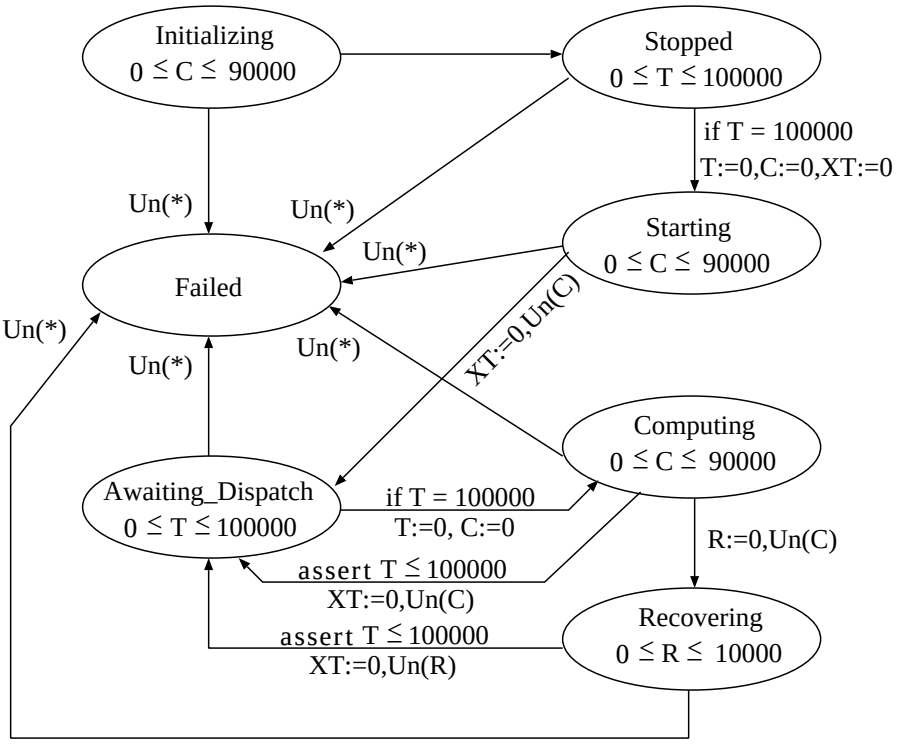


Figure 7: Generated Hybrid Automata Model for a Simple Periodic Task

niques such as testing, but it is at least intuitively reasonably easy to distinguish requirements that will be verified using hybrid automata from requirements that must be verified using other techniques.

6 Future Work

Our experience leads us to believe that linear hybrid automata are very powerful and well-suited for this domain. We were able to achieve one of our goals, the modeling and verification of a piece of real-world real-time software, with a number of limitations. We do not believe we have achieved the other goal yet, modeling and schedulability analysis for complex distributed systems of real-world size. However, there are a number of potential future developments that might reduce the verification limitations and provide useful schedulability analysis capabilities.

It should be possible to use the set of reachable regions produced by the analysis tool to automatically generate tests. This could significantly reduce the cost and increase the quality of requirements testing (which might still be required by the powers-that-be). Such tests could also detect defects that could not be found by model analysis, such as defects in the compiler, linker, loader, RTOS or hardware. One of the issues that must be confronted is the ease of constructing, running and observing the results of tests; for example, in theory one might encounter transitions in the model that occur only when two values are extremely close, which could be practically impossible to do in a test. Another issue is that such tests would not take into account the internal logic of unmodeled modules such as the RTOS; a systematic method for testing multiple points within each reachable polyhedron might help address this.

There are a number of potentially useful improvements in analysis methods and tools. Approximation and partial order methods might significantly increase the size of the model that could be analyzed[16, 19, 15, 29]. Preprocessing models to modify numeric parameters in certain ways can result in much more easily solved models[29]. It is possible to apply theorem proving methods to linear hybrid automata[21], and some work has been done on dense-time process algebras[10, 14]. Decomposition and induction methods currently being explored for discrete state models might be extensible to linear hybrid automata. There are a number of possible ways to visualize and navigate the reachable region space that would be of practical assistance during model development and debugging and during reviews. Concise APIs and support for inline modeling could reduce both the modeling effort and the number of modeling defects.

Changes will inevitably be required to the design, implementation and verification processes to make good use of these methods. Much of the benefit of other formal methods has been due to subsequent changes in development methods that resulted in more verifiable and defect-free specifications, designs and code in the first place. An important and not completely technical question is how verification processes might be changed to beneficially use these methods.

Description	Discrete States	Distinct Polyhedra	Sparc Ultra-2 CPU Seconds
one periodic task	7	7	0
one periodic task, enforced execution time limits	7	10	0
one periodic task, enforced execution time limits, one semaphore	8	29	15
one period-enforced aperiodic task	9	18	0
one period-enforced aperiodic task, enforced execution time limits	9	27	2
one period-enforced aperiodic task, enforced execution time limits, one semaphore	11	124	125
two periodic tasks	36	60	3
two periodic tasks, enforced execution time limits	36	108	24
two periodic tasks, one with period transformed into two pieces,	41	97	10
two periodic tasks, one shared semaphore	48	118	36
two periodic tasks, one with period transformed into two pieces, enforced execution time limits	41	174	87
two periodic tasks, one with period transformed into four pieces, enforced execution time limits, recovery limit greater than compute limit	40	334	103
two tasks, one periodic and one period-enforced aperiodic	44	623	115
two periodic tasks, one with period transformed into four pieces, enforced execution time limits	41	351	170
two tasks, one periodic and one period-enforced aperiodic, enforced execution time limits	44	425	184
two tasks, one periodic and one period-enforced aperiodic, one shared semaphore	70	638	840
two periodic tasks, one with period transformed into two pieces, enforced execution time limits, one shared semaphore	55	963	5658

Table 1: Modeled Applications

What evidence would be required, for example, to convince a development organization or regulatory authority to replace selected existing verification activities with modeling and analysis activities, or to add modeling and analysis to current verification activities?

References

[1] *MetaH User's Guide*, Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN, `www.htc.honeywell.com/metah`.

[2] K. Altisen, G. GöBler, A. Pnueli, J. Sifakis, S. Tripakis and S. Yovine, "A Framework for Scheduler Synthesis," *Real-Time Systems Symposium*, December 1999.

[3] Rajeev Alur, Tomás Feder and Thomas A. Henzinger, "The Benefits of Relaxing Punctuality," *Proceedings of the Tenth Annual ACm Symposium on Principles of Distributed Computing*, Montreal, Quebec, August 19-21, 1991.

[4] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho, "Automatic Symbolic Verification of Embedded Systems," *IEEE Transactions on Software Engineering*, vol. 22, no. 3, March 1996, pp 181-201.

[5] Pam Binns, "Scheduling Slack in MetaH," *Real-Time Systems Symposium*, work-in-progress session, December 1996.

[6] Pam Binns, "Incremental Rate Monotonic Scheduling for Improved Control System Performance," *Real-Time Applications Symposium*, 1997.

[7] Pam Binns and Steve Vestal, "Message Passing in MetaH using Precedence-Constrained Multi-Criticality Preemptive Fixed Priority Scheduling," *submitted Real-Time Applications Symposium.*

[8] Johan Bengtsson and Fredrik Larsson, *UPPAAL, A Tool for Automatic Verification of Real-Time Systems*, DoCS 96/97, Department of Computer Science, Uppsala University, January 15, 1996.

[9] B. A. Brandin and W. M. Wonham, "Supervisory Control of Timed Discrete-Event Systems," *IEEE Transitions on Automatic Control*, v39, n2, February 1994.

[10] Patrice Brémond-Grégoire and Insup Lee, "A Process Algebra of Communicating Shared Resources with Dense Time and Priorities," University of Pennsylvania Department of Computer Science Technical Report MS-CIS-95-08, June 1996.

[11] S. Campos, E. Clarke, W. Marrero, M. Minea and H. Hiraishi, "Computing Quantitative Characteristics of Finite-State Real-Time Systems," *Real-Time Systems Symposium*, December 1994.

[12] David L. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *International Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, June 12-14, 1989, also in *Lecture Notes in Computer Science 407*, J. Sifakis (Ed.), Springer-Verlag, pp 197-212.

[13] Andre N. Fredette and Rance Cleaveland, "RSTL: A Language for Real-Time Schedulability Analysis," *Proceedings of the Real-Time Systems Symposium*, December 1993.

[14] Andre N. Fredette, *A Generalized Approach to the Analysis of Real-Time Computer Systems*, Ph.D. Dissertation, North Carolina State University, March 1993.

[15] Nicolas Halbwachs, Pascal Raymond and Yann-Eric Proy, "Verification of Linear Hybrid Systems by Means of Convex Approximations," *Workshop on Verification and Control of Hybrid Systems*, Piscataway, NJ, October 1995.

[16] Nicolas Halbwachs, Yann-Erik Proy and Patrick Roumanoff, "Verification of Real-Time Systems using Linear Relation Analysis," *Formal Methods in System Design*, 11(2):157-185, August 1997.

[17] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri and Pravin Varaiya, "What's Decideable About Hybrid Automata?" *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 1995.

[18] Thomas A. Henzinger, Pei-Hsin Ho and Howard Wong-Toi, "HyTech: The Next Generation," *Real-Time Systems Symposium*, December 1995.

[19] Thomas A. Henzinger and Pei-Hsin Ho, "A Note On Abstract Interpretation Strategies for Hybrid Automata," *Hybrid Systems II*, also *Lecture Notes in Computer Science 999*, Springer-Verlag, 1995.

[20] Thomas A. Henzinger, Pei-Hsin Ho and Howard Wong-Toi, "A User Guide to HyTech," University of California at Berkeley, `www.eecs.berkeley.edu/~tah/HyTech`

[21] Thomas A. Henzinger and Vlad Rusu, "Reachability Verification for Hybrid Automata," *Proceedings of the First International Workshop on Hybrid Systems: Computation and Control*, also *Lecture Notes in Computer 1386*, Springer-Verlag, 1998.

[22] Y. Kesten, A. Pnueli, J. Sifakis and S. Yovine, "Integration Graphs: A Class of Decideable Hybrid Systems," in R. L. Grossman, A. Nerode, A. P. Ravn and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, Springer-Verlag, 1993.

[23] Insup Lee, Patrice Brémond-Grégoire and Richard Gerber, "A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems," Department of Computer Science, University of Pennsylvania.

[24] Bruce Lewis, "Software Portability Gains Realized with MetaH, an Avionics Architecture Description Language," 18th *Digital Avionics Systems Conference*, St. Louis, MO, October 24-29, 1999.

[25] Anum Puri and Pravin Varaiya, "Decidability of Hybrid Systems with Rectangular Differential Inclusions," Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA.

[26] Peter J. G. Ramadge and W. Murray Wonham, "The Control of Discrete Event Systems," *Proceedings of the IEEE*, v77, n1, January 1989.

[27] Steve Vestal, "An Architectural Approach for Integrating Real-Time Systems," *Workshop on Languages, Compilers and Tools for Real-Time Systems*, June 1997.

[28] Steve Vestal, "Linear Hybrid Automata Models of Real-Time Scheduling and Allocation in Distributed Heterogeneous Systems," Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN 55418, 1999.

[29] Steve Vestal, "A New Linear Hybrid Automata Reachability Procedure," Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN 55418, 1999.

[30] Steve Vestal, "Formal Verification of the MetaH Executive Using Linear Hybrid Automata," Honeywell Technology Center, Minneapolis, MN 55418, December 1999.

[31] Jin Yang, Aloysius K. Mok and Farn Wang, "Symbolic Model Checking for Event-Driven Real-Time Systems," *ACM Transactions on Programming Languages and Systems*, v19, n2, March 1997.